8주차 예비보고서

전공: 수학/컴퓨터공학 학년: 3학년 학번: 20181294 이름: 임승섭

**1.**

1 - 1. 추천 기능

추천 시스템에서는 현재 블록(nextblock[0])과 다음 두 블록(nextblock[1], nextblock[2])들의 모든 조합을 고려하여 가장 높은 점수를 얻을 수 있는 위치를 알려준다. 각 블록들은 블록의 모양에 따라, 블록의 회전수에 따라 서로 다른 경우의 수를 가진다. 이 시스템은 Tree 구조를 이용한다.

Tree의 level 1에는 현재 블록의 정보가 저장되고, level 2와 3에 각각 다음 블록의 정보가 저장된다. 즉, 하나의 level에는 모두 같은 모양의 블록의 정보가 저장되게 된다. 같은 레벨이지만 노드별로 다른 점은 블록의 위치와 회전수가 된다. 따라서 하나의 노드는 블록의 ID(curBlockID), 블록의 위치(recBlockX, recBlockY), 블록의 회전수(recBlockRotate), 필드의 상태(recField)(쌓일 예정인 블록도 계산하기 때문에 Field와는 다르다), 얻게 되는 점수 (accumulatedScore)로 구성된다.

이렇게 노드를 구성하면 현재 블록(nextblock[0])에 대한 모든 경우의 수를 level 1의 노드들로 파악할 수 있다. 같은 방법으로 다음 블록(nextblock[1])에 대한 경우의 수도 모두 파악할 수 있다. 그 노드들을 level 1에 있는 모든 노드들의 child로 설정해주면 된다. 그렇게 되면 level 2에는 총 nextblock[0]의 가능한 경우의 수 \* nextblock[1]의 가능한 경우의 수 만큼의 노드가 생기게 된다. 마찬가지의 방법으로 level 3도 구성한다.

총 3개의 level로 tree를 구성하였다면, level 3의 노드들 중 가장 큰 accumaulatedScore를 가진 노드를 찾고, Root node에서 그 노드로 가기 위한 level 1의 노드를 찾는다. 이 노드를 사용자에게 추천한다.

1 - 2. tree 구조를 이용한 추천 기능의 장점

점수를 가장 잘 받기 위한 정보를 알려주기 때문에 고득점을 획득하는 데 유리하다. 현재 위에서는 미래의 두 블록에 대한 정보를 파악하고 level 2의 tree 구조를 만들었지만, 블록의 수가 많아질수록 더 높은 점수를 얻을 수 있게 된다. 확실하게 알고 있는 미래의 범위가 많아질수록 현재의 선택이 효율적이 되는 것은 자명하다.

1 - 3. tree 구조를 이용한 추천 기능의 단점

다음 블록을 많이 고려할수록 높은 점수는 받을 수 있지만, 그에 따른 시간과 공간을 소모하는 면으로 보았을 때는 매우 비효율적이다.

우선, 시간 측면에서 보았을 때, 현재 시스템에서는 각 level의 모든 노드를 거쳐 점수를 확인해야 한다. 하나의 블록이 가능한 경우의 수가 n개라고 하고, 모두 같은 경우의 수를 갖는다고 가정했을 때, level 1의 모든 노드의 개수는 n개, level 2의 모든 노드의 개수는 n2개이다. 결국 level m까지 있을 때, 이 트리의 leaf node의 개수는 nm개가 된다. 하나의 노드를 찾기 위해 이처럼 exponentail로 급증하는 노드들을 모두 확인하는 것은 시간 측면에서 매우 비효율적이다. asymtotic한 time complexity가 exponential로 증가하기 때문에, 시간이 증가하는 양이 점점 더 커진다.

또한 공간 측면에서 보았을 때, tree의 모든 노드들은 하나의 공간을 차지하고 있고, 그 하나의 공간에는 블록의 id(curBlockID), 블록의 위치(recBlockX, recBlockY), 블록의 회전수(recBlockRotate), 필드 정보(recField), 누적 점수(accumulatedScore), 자식 노드의 정보 등 많은 공간을 또 차지하고 있다. 하나의 노드에 필요한 메모리적 공간을 k라고 하고, 하나의 블록이 가능한 경우의 수가 n개, level m까지 총 m개의 노드를 고려한다고 가정하면, 필요한 메모리 공간은 k\*nm이 된다. 위와 마찬가지로 exponential하게 급증하고 있는 형태이기 때문에, 원하는 노드 하나를 찾기 위해 이렇게까지 많은 메모리 공간을 사용하는 것은 아주 비효율적이다.

**2.**

2 - 1. level의 개수를 줄인다.

시간과 공간의 효율성을 낮추는 가장 직접적인 이유는 파악하고자 하는 미래의 블록들이 많기 때문이다. 직관적으로 생각했을 때 미래 블록들을 줄이면, 효율성이 올라간다. 현재 위에서 tree를 level 3로 만들었다면, level 2까지만 tree를 구성하여 효율성을 높인다. 하지만, 그만큼 높은 점수를 받기 위한 전략으로는 좋지 않다. 다음 노드의 경우의 수만 고려하고 정확한 예측을 하긴 쉽지 않기 때문이다.

2 - 2. pruning

강의자료에 있는 pruning 방식을 이용한다. pruning이란 가지를 잘라 주는 일을 뜻한다. tree에서 각 노드로 뻗쳐 있는 branch를 제거한다. 필요 없는 노드들을 제거함으로써, 그 아래에 있는 노드들까지 고려할 필요가 없어지기 때문에 시간/공간적으로 아주 효율적인 알고리즘이 될 수 있다. 문제는 노드를 제거하는 기준인데, 이 기준에 따라 또 결과가 크게 달라질 수 있다. 가장 낮은 점수를 갖고 있는 노드 한 개만 줄이게 되면, maximum score로 갈 수 있는 확률이 높아지지만, 효율성이 크게 향상되진 않는다. 만약 가장 높은 점수를 갖는 노드 한 개만 살리게 되면, 효율성은 급격하게 높아지지만, maximum score로 갈 수 있는 확률은 현저하게 낮아진다. 상위 level에서 score가 높다고 해서, 최종 score가 maximum이라는 보장이 없기 때문이다. 따라서 특정 점수로 기준을 잡아, 그 점수보다 아래에 있는 노드들은 생성하지 않는 것이 가장 좋은 방법이라고 생각한다.

2 - 3. 데이터의 단순화

강의자료에 있는 데이터의 단순화 방식을 이용한다. 현재 tree 구조의 모든 노드에는 field의 정보가 들어있다. 이 정보가 공간을 가장 많이 소모하고 있기 때문에, field의 정보를 단순화시키면 된다. 모든 x좌표에서 가장 높이 채워져 있는 y좌표를 확인하고 그 밑에를 다 차있다고 가정한다. 이후, 블록이 쌓일 때에는 그 높이만을 가정하고 점수를 계산한다. 이렇게 되면 필드의 정보에 대한 공간을 상당히 많이 확 줄일 수가 있고, 결국 공간적 효율성을 증가시킨다. 하지만 점수를 계산할 때, 아래에 빈 공간이 있는 걸 확인할 수 없기 때문에 계속 deleteline을 할 수 있고, 전혀 엉뚱한 높은 점수를 출력해 낼 수도 있다. 따라서 이를 보완하기 위해서는, 맨 위부터 4칸 아래까지만 정확한 field 정보를 확인하고, 그 아래는 똑같이 모두 채워져 있다고 가정하는 방법이 있다. 블록 중 가장 높은 블록은 1x4 크기의 1자 형태의 블록이기 때문에, 블록들이 field의 한 줄을 제거할 수 있는 범위는 최대 4 높이이다. 따라서 모든 x좌표에 대해 가장 높은 y좌표에서 아래로 4칸까지만 알고 있으면 효율성은 낮아지지만, 정확성은 높아질 수 있다. 하지만 이 방법도 문제가 있다. x좌표별로 높이가 너무 크게 다르면 정확성 면에서 또 문제가 발생한다. 이마저도 보완하기 위해서는 모든 x좌표에서 가장 낮은 높이를 확인하고 여기서부터 아래로 4칸을 모든 x에 대해 확인하는 것이다. 사실 이렇게 하면 field에 블록이 아주 많이 채워져 있지 않는 한 효율성을 크게 높이긴 어렵다. 그래도 field가 많이 채워져있어서 y좌표가 꽤 높을 때는, 정확성과 효율성을 어느 정도 보장할 수 있는 방법이다.